УДК 004.41:004.42

ФАКТОРИ ЕФЕКТИВНОСТІ МЕТОДІВ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В. М. Сеньківський, О. З. Білик, Н. Є. Сеньківська

Українська академія друкарства,
вул. Під Голоском, 19, Львів, 79020, Україна

Розроблення програмного забезпечення — складний багатоетапний процес. Структуризація та управління цими етапами значною мірою впливає на якість кінцевого продукту, часові рамки проєкту та ефективність залучення робочої сили. На сьогодні існує низка методологій для організації процесу розроблення програмного забезпечення. Кожна із методологій створена з метою задоволення конкретних потреб проєкту і, отже, має сильні та слабкі сторони. На жаль, через брак кількісних даних і порівняльних характеристик різних методологій багато компаній розробників часто обирають найпопулярнішу з них, незалежно від того, наскільки вона придатна для певного типу програмного забезпечення, яке вони створюють.

У пропонованому дослідженні виконано огляд декількох сучасних методологій, дотичних до процесу розроблення програмного забезпечення. Виокремлено та проведено порівняльний аналіз множини ключових факторів, що впливають на рішення щодо вибору найбільш відповідної методології для конкретного проєкту зі створення програмного продукту.

Ключові слова: *програмне забезпечення, методологія розроблення програмного забезпечення, фактори вибору методології, життєвий цикл, основні характеристики методологій.*

Постановка проблеми. Основною метою цього дослідження є виявлення та аналіз ключових факторів, які впливають на прийняття рішення щодо вибору найбільш доцільної методології розроблення програмного забезпечення (ПЗ) для конкретного проєкту. Ознайомлення з найпоширенішими засобами розроблення ПЗ дає підставу стверджувати, що одні методології орієнтовані на процес, інші на людей, деякі спеціалізуються на певному типі продукту, інші є більш загальними, певні з них придатні для малих програмних систем, інші ефективні лише для великих проєктів тощо. Оскільки зміна методології після її впровадження займає багато часу та витрат для компанії, тому важливо, щоб вибір оптимального засобу розроблення ПЗ був завершений стадією формування та затвердження технічного завдання на проєкт.

Аналіз останніх досліджень та публікацій. У науковій літературі доступний широкий спектр досліджень тематики, пов’язаної з еволюцією методологій розроблення програмного забезпечення. На ранніх етапах виникнення вказаних

методологій планування та реалізація означеного процесу здійснювалися за лінійною, послідовною схемою. Один із перших «традиційних» стандартів був офіційно представлений у статті, що написав Вінстон В. Ройс у 1970 році, пізніше отримавши назву Waterfall [1]. Ця методологія була успішною лише для розроблення систем, вимоги до яких чітко визначалися на початковому етапі проєкту. Відповідно, вона була піддана критиці, як така, що не враховує динамічну природу розроблення [2, 3]. У 2001 році було запропоновано низку альтернативних підходів, об'єднавчими характеристиками яких є поетапне визначення вимог і тісна кооперація клієнта та розробника [4]. Вказані підходи також більш відомі в літературі як «гнучкі» методології. Для дослідження у цій статті, крім згаданої вище методології Waterfall, обрано підходи RAD (Rapid Application Development) [5] та Scrum [6], що посіли значне місце у сфері розроблення програмного забезпечення. Причиною такого вибору стала властивість RAD, що поєднує в собі елементи як «традиційної», так і «гнучкої» методології, в той час як Scrum є яскравим представником саме «гнучкого» підходу. Про переваги цих методологій свідчать численні згадки у низці досліджень [7−11].

Попри велику кількість публікацій із описом переваг та недоліків кожної із методологій, більшість з них не акцентують уваги на виокремленні основних факторів, що зумовили б вибір найбільш ефективної для конкретної системи. Наприклад, у статті [12] викладено розлогий аналіз підходів розроблення ПЗ, проте виділено лише один фактор. У публікаціях [13, 14] проведено аналітику емпіричних даних щодо вибору методу планування та визначальних факторів, які впливають на фінальне рішення щодо майбутнього програмного забезпечення. У розвідці [15] наведено результати детального дослідження факторів, що впливають на процес розроблення програмного забезпечення. Однак згадані праці не визначають зв'язку між факторами та методологіями і, відповідно, не можуть бути використані для вибору задовільного підходу. Найбільш близькі результати, дотичні до нашого дослідження, розглянуто у публікації [11].

З огляду на сказане, дослідження, яке ми пропонуємо, має на меті проаналізувати наведені вище методології розроблення програмного забезпечення, виокремити та розглянути спільну для всіх множину факторів, що впливають на вибір методології та її придатність забезпечити належну якість програмного продукту, отриманого з використанням вказаних засобів.

Мета статті — виокремити та виконати аналітичний огляд факторів, за якими можна оцінити засоби розроблення програмного забезпечення на прикладі методологій Waterfall, Rapid Application Development (RAD) та Scrum.

Виклад основного матеріалу дослідження. У статті розглянуто методології розроблення програмного забезпечення Waterfall, Rapid Application Development (RAD) та Scrum. Для кожної з методологій виділено основні характеристики, розглянуто модель життєвого циклу та проведено аналіз сильних і слабких сторін.

Waterfall. Методологія *Waterfall* (Водоспад) використовує послідовний або лінійний підхід до розроблення програмного забезпечення. Проєкт розбивається на послідовність завдань, які називаються фазами. Вказана методологія потребує,

щоб фази завершувалися послідовно і мали формальні критерії завершення, зазвичай, підписання зацікавленими сторонами проєкту.

Типовий перелік завдань методології *Waterfall* може містити:

– визначення обсягу та планування проєкту;
– збір та документування вимог;
– розроблення проєктної архітектури;
– розроблення програми та написання модульних/інтеграційних тестів;
– проведення тестування;
– виправлення помилок за необхідності та розгортання програми.

Методологія *Waterfall* є формальним процесом, кожна фаза якого складається з переліку детальних завдань із супровідною документацією, виконання яких можливе лише після завершення попередньої фази (рис. 1).

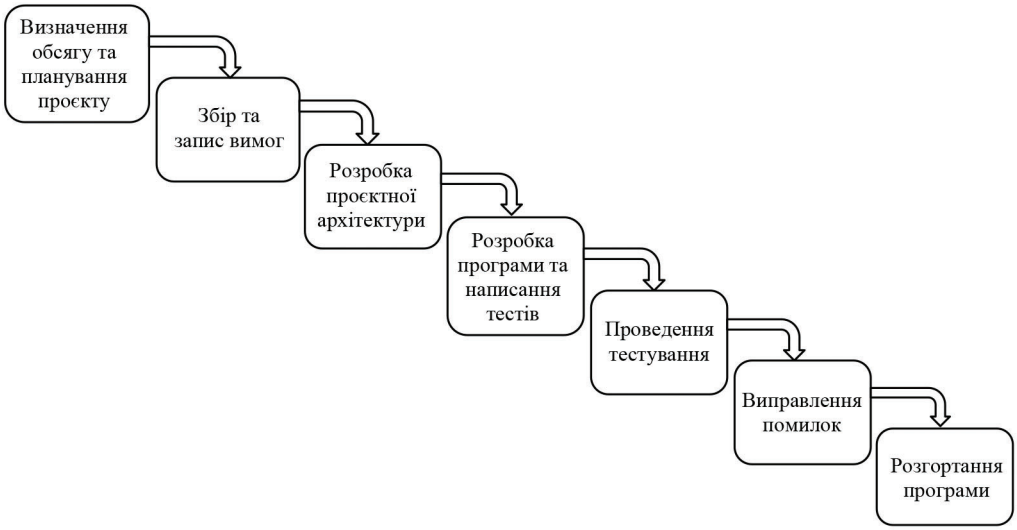

Рис. 1. Життєвий цикл розроблення програмного забезпечення при Waterfall підході

Переваги методології Waterfall:

– наявність чітко розробленої схеми процесу розроблення ПЗ для виконання кожного етапу проєкту;
– вимоги формуються на ранній стадії, що дає змогу команді визначити весь обсяг проєкту, сформувати графік і розробити загальну архітектуру продукту;
– зменшується ймовірність «поганої» комунікації в команді, оскільки ролі кожного учасника заздалегідь визначені;
– чітко сформовані вимоги допомагають створювати ґрунтовну документацію.

Недоліки методології Waterfall:

– для великої кількості проєктів важко отримати повні вимоги від клієнтів перед початком розроблення;
– на початковому етапі необхідне залучення досвідчених спеціалістів для детального вивчення та розподілу завдань і етапів роботи між виконавцями;

– звичайно, проєкти, які використовують Waterfall, за своєю суттю не мають охоплювати значні проміжки часу, однак на практиці дуже часто вони займають місяці або квартали через те, що акцент робиться на спробі зробити все відразу. Відповідно, значними стають ймовірність затримки, перевищення бюджету, невиконання очікувань замовника, що сукупно накопичуються у випадку, коли терміни реалізації проєкту значно збільшуються.

RAD. Методологія *RAD* дає змогу швидко розробляти інформаційні системи з відносно низькими витратами. У цьому підході програмне забезпечення поділяється на малі компоненти (модулі), що полегшує внесення змін протягом усього періоду розроблення. Для окремих компонент проєкту встановлюються визначені терміни виконання, що не мають бути перевищені.

Методологія *RAD* може використовувати інші методи, такі як *JAD*, спіральний процес розроблення або прототипування. У проєктах *RAD* модулі, що відображаються під час прототипування, стають складовими програмного забезпечення. Також, як і в інших методологіях, є можливість повторного використання компонент.

На відміну від структурованих методологій, які передбачають охоплення великої кількості кроків для отримання програмного продукту, методологія *RAD* пропонує декілька кроків, у ході яких активно залучають команду розробників та користувачів, що призводить до більш швидкого отримання готового програмного продукту. Життєвий цикл програмного забезпечення *RAD* складається з чотирьох етапів: формування вимог, проєктування, реалізація, розгортання (рис. 2).

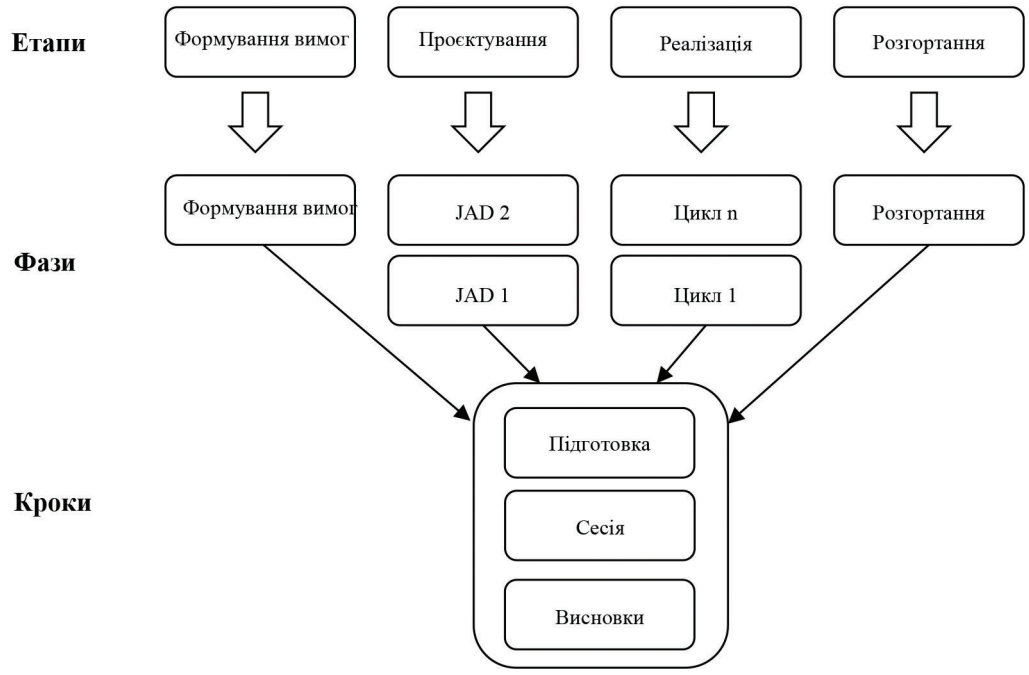

Рис. 2. Життєвий цикл розроблення програмного забезпечення при RAD підході

Кожний етап передбачає виконання однієї або декількох фаз. Окрема фаза процесу розроблення ПЗ містить три кроки.

Підготовка. Підготовка матеріалів для обговорення під час сесії.

Сесія. Учасники процесу розроблення програмного забезпечення зустрічаються для ухвалення рішень про те, як повинен реалізуватися той чи інший етап життєвого циклу.

Висновок. Результат сесії формулюється у вигляді висновків, які будуть враховані у процесі розроблення ПЗ.

У проєктах, що розробляються за методологією *RAD*, обов'язки чітко розподілені між різними учасниками. Так, кожен учасник може виконувати одну з ролей, що передбачено вказаною методологією — менеджер проєкту, користувач, *RAD*-експерт, розробник прототипів, власник. Методологія *RAD* концентрується на користувачах, що активно залучаються до процесу розроблення.

Методологія *RAD* має значні переваги, порівняно з методологіями, що використовувалися до її появи. Використання *RAD* уможливило скорочення часу, необхідного для отримання кінцевої системи, зниження витрат і зменшення ризику через введення користувачів в команду розробників. Використання прототипів *RAD* дає змогу клієнтам взаємодіяти з варіантами системи вже на ранніх етапах процесу розроблення. Отже, вимоги, що змінюються, можуть бути швидко внесені в систему.

Однак існують певні ризики під час впровадження методології *RAD*, пов'язані з виконанням початкових вимог. Оскільки зазвичай вони не розглядаються систематично, а команда працює швидко за допомогою ітерацій, тому виникає ймовірність упущення деяких важливих аспектів продукту. Потрібно також зауважити, що методологія *RAD* інколи нехтує аспектами, пов'язаними з управлінням системами (підтримка та реорганізація баз даних, резервне копіювання, відновлення після збоїв системи тощо).

SCRUM. Методологія *Scrum* базується на наборі чітко визначених практик та ролей, які мають бути задіяні у процесі розроблення програмного забезпечення. Вона передбачає застосування 12 принципів гнучкого управління в контексті, узгодженому з усіма членами команди, які працюють над продуктом. *Scrum* використовує короткі ітерації, так звані спринти. Вони зазвичай складають від 2 до 4 тижнів. В кінці кожного спринту клієнт повинен мати змогу отримати певний завершений функціонал, який він зможе оцінити, дати відгук та, якщо буде потреба, внести правки (рис. 3). Відправною точкою *Scrum* є перелік цілей/вимог, які становлять план проєкту. Замовник проєкту визначає їх пріоритетність, базуючись на вартості та важливості для продукту.

Scrum-команда складається з таких ролей:

– *Scrum Master* — особа, яка керує командою, стежить за дотриманням правил і процесів методології та допомагає вирішувати питання, які виникають у процесі розроблення. Також в його обов'язки входить моніторинг показників продуктивності команди та дотримання термінів виконання проєкту.

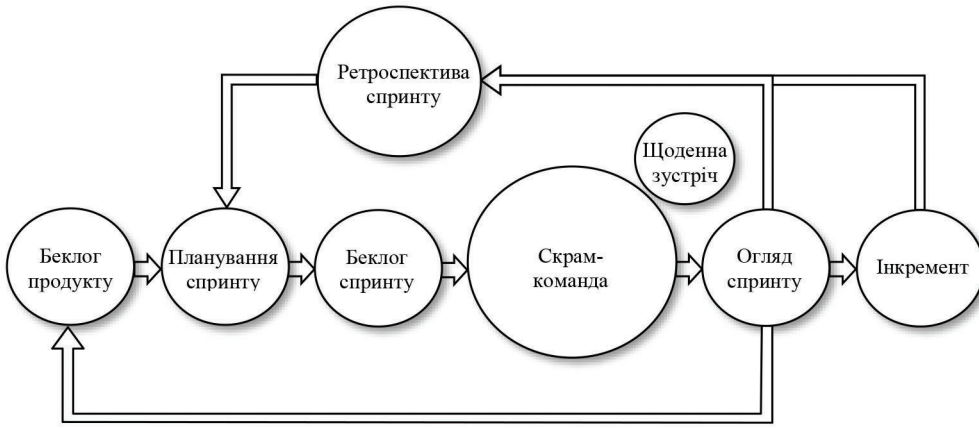

Рис. 3. Життєвий цикл програмного забезпечення при SCRUM підході

– *власник продукту* — представник зацікавлених сторін та клієнтів, які використовують програмне забезпечення. Він зосереджується на бізнес-частині і відповідає за рентабельність інвестицій у проєкт. Власник продукту передає бачення проєкту команді, формує завдання і регулярно визначає їх пріоритетність.
– *команда* — група професіоналів з необхідними технічними знаннями, які спільно розробляють проєкт, виконуючи завдання, які вони беруть на себе на початку кожного спринту.

Події в Scrum:

– *спринт* (*Sprint)* — основна одиниця роботи Scrum-команди. Всередині спринту виконується вся робота, необхідна для досягнення цілі продукту, зокрема планування спринту, щоденні зустрічі, огляд і ретроспектива спринту.
– *планування спринту (Sprint planning)* — визначення того, що і як буде зроблено в спринті. Планування проводиться на початку кожного спринту.
– *щоденні зустрічі* (*Daily scrum)*. На щоденних зустрічах відбувається оцінка прогресу роботи, синхронізація діяльності та створення плану на наступний робочий день. Кожен учасник команди дає відповіді на три запитання: Що я робив вчора? Що я збираюся робити сьогодні? Яка допомога мені потрібна? Scrum Master має допомогти вирішити проблеми або усунути перешкоди в роботі, які виникають у членів команди.
– *огляд спринту (Sprint review)* — це подія, яка відбувається в кінці спринту і передбачає демонстрацію функціонуючого продукту, розробленого протягом спринту. Головною метою огляду спринту є детальне представлення досягнень команди перед клієнтами.
– *ретроспектива спринту (Sprint retrospective)*. Команда переглядає виконані задачі завершеного спринту, робить аналіз помилок і документує хороші та погані тенденції. Мета ретроспективи — виявити можливі покращення процесу розроблення і сформувати план їх реалізації в наступному спринті.

Scrum артефакти:

– *беклог продукту* (*Product backlog)* — всі необхідні дії, пов'язані з користувацькою і технічною сторонами проєкту.
– *беклог спринту (Sprint backlog)* — це сукупність усіх завдань, які потрібно виконати протягом однієї ітерації спринту. Його складають із задач беклогу продукту під час планування спринту.
– *інкремент (Increment)* — це сукупність всіх завдань з беклогу продукту, які успішно були виконані протягом спринту, а також усіх інкрементів з попередніх ітерацій. Перед закінченням спринту новий інкремент має бути працездатним та відповідати зазначеним раніше критеріям готовності.

Методологія *Scrum* має такі переваги:

– *Передбачуваність.* Scrum зосереджується на плануванні, що можна трактувати як перевагу, так і недолік. Зокрема, це особливо корисно, коли необхідно ефективно керувати випуском різних компонентів програмного забезпечення та забезпечувати їх відповідність конкретним бізнес-цілям на регулярній основі, наприклад, щокварталу. Додатковою вимогою Scrum є завершеність задач під час спринту. Тобто коли спринт закінчиться, будуть отримані конкретні результати.
– *Швидке удосконалення продукту.* Scrum забезпечує швидкі ітерації та покращення. Оскільки певні частини продукту можуть бути випущені незалежно, їх можна тестувати і аналізувати відразу. Це дає можливість без затримки вносити правки в наступному спринті.
– *Краща взаємодія в колективі.* Scrum передбачає активну комунікацію та співпрацю між учасниками команди з метою успішного виконання завдань протягом визначеного часу спринту.
– *Самоорганізація.* Самоорганізація є однією з ключових рис Scrum команди, оскільки вона дає змогу команді керувати своїм часом та роботою в межах спринту без необхідності управлінського нагляду. Це дає змогу команді ухвалювати рішення швидше та ефективніше.

Недоліками методології *Scrum* вважаються:

– *Необхідність структурування робочого процесу.* В Scrum багато зустрічей і конкретних процесів, тому його не так легко налаштовувати на різні бізнес-сценарії. Команді доведеться звикати та підлаштовуватись до наявності конкретних ролей (Scrum Master, власник продукту тощо) та подій (щоденні зустрічі, огляди спринтів). Крім того, через наявність чітко визначених часових рамок неможливо додавати або скасовувати завдання під час проведення спринту.
– *Забагато нарад.* Незважаючи на те, що кожен спринт є короткочасним, він все ж потребує кількох зустрічей для обговорення прогресу роботи. Якщо зустрічі проводять занадто часто або вони тривають довше, ніж потрібно, це може вплинути на продуктивність команди, адже цей час запланований на виконання завдань. Щоденні наради мають бути відносно короткими — не більше 15 хвилин. Таким чином, команда матиме можливість спланувати роботу на день.

– *Scrum потребує часу на впровадження.* Для того щоб співробітники зрозуміли, як ефективно працювати над проєктом, може знадобитись декілька спринтів.
– *Необхідно правильно планувати робоче навантаження.* Щоб уникнути затримок в проєкті, потрібно правильно розбити роботу на менші компоненти та дбайливо спланувати навантаження команди на кожен спринт відповідно до Scrum.

Аналіз факторів ефективності методів розроблення програмного забезпечення.

Придатність методології розроблення ПЗ для певного проєкту залежить від низки факторів. Серед найважливіших можна виділити такі: чіткість початкових вимог, початкова оцінка витрат і часу розроблення, зміни вимог у процесі розроблення, отримання функціональних версій системи під час розроблення, витрати на розробку, тривалість часу доставки кінцевої версії системи, складність системи, контакти між клієнтами та розробниками. Описані вище методології (Waterfall, RAD та SCRUM) будемо аналізувати відповідно до цих факторів.

1. Чіткість початкових вимог

Waterfall. Початкові вимоги мають повністю описувати функціональні можливості кінцевої системи, оскільки на їх основі розробляється документація, дизайн та архітектура. Waterfall працює за каскадним принципом і не має ітерацій, тому будь-які зміни у процесі розроблення практично неможливі.

RAD. Ця методологія містить змінну кількість циклів створення прототипів і полягає у покроковому формуванні життєздатного програмного забезпечення. Повні функціональні вимоги не встановлюються на початку проєкту, їх детально вказують користувачі на кожній ітерації.

SCRUM. Оскільки ця методологія складається з коротких циклів і вимоги можуть змінюватися на кожній ітерації, то не обов'язково, щоб вони були повністю описані перед початком розроблення системи.

2. Початкова оцінка витрат і часу розроблення

Waterfall визначає чіткі етапи розроблення та передбачає точні початкові вимоги до продукту. Маючи ці дані, можна з певною похибкою дати оцінку часу та витратам.

RAD. Для кожного з п'яти етапів компонент методології RAD встановлюється максимальна кількість днів для досягнення цілей етапу. Залежно від специфіки кожного проєкту ефективний час розроблення можна оцінити з невеликою похибкою. Початкова оцінка витрат на розробку може змінюватися залежно від зусиль, витрачених на впровадження вимог, які змінюються у процесі розроблення.

SCRUM. Важко оцінити зусилля, необхідні для розроблення всієї системи, оскільки не всі вимоги відомі на початку проєкту.

3. Зміни вимог у процесі розроблення

Waterfall. Ця методологія дуже негнучка щодо зміни вимог під час процесу розроблення. Внесення суттєвих змін означатиме, що цілий процес потрібно розпочати спочатку, адаптуючи документацію, дизайн та архітектуру.

RAD. Інформаційна система розділена на малі сегменти, що полегшує внесення змін у процесі розроблення в будь-який час циклу.

SCRUM. Методологія є гнучкою і легко адаптується до змін вимог. Зміни системи можуть бути внесені навіть на пізніх стадіях життєвого циклу.

4. Отримання функціональних версій системи у процесі розроблення

Waterfall. Клієнт не отримує робочу версію продукту, доки не завершиться весь цикл розроблення.

RAD дає змогу користувачам взаємодіяти з варіантами системи на ранніх стадіях завдяки використанню прототипів.

SCRUM передбачає часте отримання замовником робочої версії розробленої системи. Такий спосіб роботи допомагає клієнтам зрозуміти прогрес у розробленні системи та дає змогу їм вносити правки.

5. Витрати на розробку

Waterfall. Витрати на розробку базуються на визначеному обсязі задач і передбачають певні процедури перед тим, як почнеться сам процес розроблення (наприклад, збір вимог, документація тощо). Відповідно, можна вважати витрати значними, порівняно з іншими методологіями, які розглядаються.

RAD. Завдяки повторному використанню прототипів і короткому часу розроблення методологія передбачає відносно низькі витрати.

SCRUM. Методологія організована в декілька коротких циклів розроблення для того, щоб зменшити вартість змін, внесених для адаптації до вимог, висловлених клієнтами протягом життєвого циклу системи. Команда розробників невелика, що передбачає низькі витрати на людські ресурси.

6. Тривалість часу доставки кінцевої версії системи

Waterfall. Замовник отримує кінцеву версію системи після завершення всіх етапів розроблення продукту.

RAD. Методологія фокусується на обмеженні часу, фіксуючи терміни завершення розроблення компонентів проєкту. Якщо є проблеми з дотриманням термінів, акцент робиться на скорочення вимог, а не на збільшення термінів. Отже, можна сказати, що метою методології RAD є надання мінімального набору необхідних вимог у найкоротший проміжок часу.

SCRUM. Методологія надає важливого значення задоволенню вимог клієнтів, передаючи програмне забезпечення за необхідності, а не у віддаленому майбутньому, коли вимоги можуть бути змінені.

7. Складність системи

Waterfall краще використовувати для малих за обсягом проєктів. Якщо виникає потреба у використанні цієї методології для великих систем, краще їх розбивати на малі підсистеми, для яких використовувати цей підхід.

RAD. Методологія може успішно використовуватись для невеликих і середніх за розміром проєктів, де чітко визначений обсяг робіт. RAD не працює ефективно, якщо використовується для розроблення складних або розподілених систем.

SCRUM. Найменшою одиницею в SCRUM є функціональна команда, яка відповідає за певний обсяг робіт. За правильного розподілу роботи і належної комунікації можна виділити потрібну кількість цих команд для реалізації продукту будь-яких розмірів.

8. Контакти між клієнтами та розробниками

Waterfall. Спілкування між клієнтами та командою розробників відбувається безпосередньо перед початком робіт і є дуже важливим етапом, який виділяють в

окремий операційний блок. Це пов'язано з тим, що після старту розроблення замовники не можуть змінювати вимоги і дуже важливо, щоб вони були зразу чітко сформовані.

RAD. Відбувається безпосередня участь замовників у процесі розроблення програмного забезпечення. Замовники беруть участь у робочих сесіях разом із розробниками, активно залучаючись до процесу визначення вимог, а також до процесу оцінювання та перевірки прототипів і кінцевого програмного забезпечення.

SCRUM. Комунікація із замовником передбачає залучення клієнтів на кожній ітерації розроблення для визначення вимог та оцінювання результату. Однак вважається, що замовник не має бути задіяний безпосередньо протягом самої ітерації.

Висновки. Якість програмного забезпечення суттєво залежить від обраної методології для його створення, а вибір оптимальної методології розроблення, що якнайкраще відповідає вимогам організації і типу проєктів, має велике значення у процесі розроблення програмних продуктів. У наведеній публікації виконано порівняння характеристик наявних підходів до створення програмного забезпечення, охарактеризовано їх переваги і недоліки з огляду ефективності застосування. Виокремлено та розглянуто спільну для всіх множину факторів, що впливають на вибір методології та її придатність забезпечити належну якість програмного продукту. Результати дослідження можуть слугувати зручним інструментом для компаній з метою вибору належного підходу до організації процесу розроблення програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Royce W. Managing the development of large software systems: Concepts and techniques. Proceedings of IEEE WESCON. 1970. Pp. 328−338.
2. Abrahamsson P., Salo O., Ronkainen J. Agile Software Development Methods - Review and Analysis. Finland, 2002. 107 p.
3. Highsmith J., Cockburn A. Agile software development: the business of innovation. IEEE Computer. 2001. Pp. 120−127.
4. Fowler M., Highsmith J. Manifesto for Agile Software Development. Software development. 2001. Pp. 1−7.
5. Martin J. Rapid Application Development. Macmillan Publishing Company, 1991. 736 p.
6. Schwaber K. Scrum Development Process. OOPSLA Business Object Design and Implementation Workshop. 1997. Pp. 117−134.
7. Beynon-Davies P., Mackay H. Rapid application development (RAD): An empirical review. *European Journal of Information Systems*. 1999. Pp. 211–223.
8. Reifer D. J. How Good Are Agile Methods? IEEE Software. 2002. Pp. 16−18.
9. Kettunen V., Kasurinen J., Taipale O., Smolander K. A study on agility and testing processes in software organizations. Proceedings of the 19th International Symposium on Software Testing and Analysis. 2010. Pp. 1−210.
10. Zhang X., Hu Tao. Software Development Methodologies, Trends, and Implications. Proceedings of the Southern Association for Information Systems Conference. 2010. Pp. 173−178.

11. Geambasu C., Jianu I. Influence Factors for the Choice of a Software Development Methodology. Accounting and Management Information Systems. 2011. Pp. 479–494.
12. Saeed S., Jhanjhi NZ. Analysis of Software Development Methodologies. *International Journal of Computing and Digital Systems*. 2019. Pp. 445–460.
13. Mahanti R., Neogi M. S. Factors Affecting the Choice of Software Life Cycle Models in the Software Industry - An Empirical Study. *Journal of Computer Science*. 2012. Pp. 1253–1262.
14. Vijayasarathy L. R., Butler C. W. Choice of Software Development Methodologies. IEEE SOFTWARE. 2016. Pp. 87–94.
15. Clarke P., O'Connor R.V. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Journal of Information Software and Technology.* 2012. Pp. 433–447.

REFERENCES

1. Royce, W. (1970). Managing the development of large software systems: Concepts and techniques. Proceedings of IEEE WESCON, 328–338 (in English).
2. Abrahamsson, P., Salo, O., & Ronkainen, J. (2002). Agile Software Development Methods - Review and Analysis. Finland (in English).
3. Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. IEEE Computer, 120–127 (in English).
4. Fowler, M., & Highsmith, J. (2001). Manifesto for Agile Software Development. Software development, 1–7 (in English).
5. Martin, J. (1991). Rapid Application Development. Macmillan Publishing Company (in English).
6. Schwaber, K. (1997). Scrum Development Process. OOPSLA Business Object Design and Implementation Workshop, 117–134 (in English).
7. Beynon-Davies, P., & Mackay, H. (1999). Rapid application development (RAD): An empirical review: European Journal of Information Systems, 211–223 (in English).
8. Reifer, D. J. (2002). How Good Are Agile Methods? IEEE Software, 16–18 (in English).
9. Kettunen, V., Kasurinen, J., Taipale, O., & Smolander, K. (2010). A study on agility and testing processes in software organizations. Proceedings of the 19th International Symposium on Software Testing and Analysis, 1–210 (in English).
10. Zhang, X., & Hu, Tao. (2010). Software Development Methodologies, Trends, and Implications. Proceedings of the Southern Association for Information Systems Conference, 173–178 (in English).
11. Geambasu, C., & Jianu, I. (2011). Influence Factors for the Choice of a Software Development Methodology. Accounting and Management Information Systems, 479–494 (in English).
12. Saeed, S., & Jhanjhi, NZ. (2019). Analysis of Software Development Methodologies: International Journal of Computing and Digital Systems, 445–460 (in English).
13. Mahanti, R., & Neogi, M. S. (2012). Factors Affecting the Choice of Software Life Cycle Models in the Software Industry - An Empirical Study: Journal of Computer Science, 1253–1262 (in English).
14. Vijayasarathy, L. R., & Butler, C. W. (2016). Choice of Software Development Methodologies. IEEE SOFTWARE, 87–94 (in English).

15. Clarke, P., & O'Connor, R.V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework: Journal of Information Software and Technology, 433−447 (in English).

doi: 10.32403/1998-6912-2023-1-66-11-22

FACTORS OF EFFICIENCY OF SOFTWARE DEVELOPMENT METHODS

V. M. Senkivskyy, O. Z. Bilyk, N. E. Senkivska

Ukrainian Academy of Printing,
19, Pid Holoskom St., Lviv, 79020, Ukraine
senk.vm@gmail.com

The end of the last century became a period of active attention to software, when the quality of software systems and individual components became active "participants" in the formation of the quality and reliability of information processes, technical and organizational complexes, in which they became one of the defining components. Since the introduction of the term "software engineering" into circulation, this issue has constantly appeared in the field of view of theoreticians and professional programmers. As a result, it has become possible to develop industrial software production, the emergence of structured programming, modularization of programs (the presence of autonomous modules and special routines) and the related development of block-oriented and object-oriented programming languages. Special attention is paid to decision-making regarding the choice of the most appropriate software development methodology for a specific project. Familiarity with the most common software development tools suggests that certain methodologies are process-oriented, others are people-oriented, some specialize in a type of product, others are more general, some are suitable for small software systems, others are effective only for large projects, and so on. Since changing the methodology after its implementation takes a lot of time and costs for the company, it is important that the choice of the optimal software development tool is the final stage of the formation and approval of the technical terms of reference for the project.

In the proposed publication, a comparison of the most characteristic software creation methodologies is made, their advantages and disadvantages are characterized in terms of their application efficiency. The set of factors common to all that influence the choice of methodology and its suitability to ensure the proper quality of the software product obtained using the specified means is singled out and considered. The results of the study can serve as a convenient tool for companies to choose the appropriate approach to the organization of the software development process.

Keywords: *software, software development methodology, main characteristics of methodologies, life cycle, methodology selection factors.*

Стаття надійшла до редакції 29.03.2023.
Received 29.03.2023.